IN 103 - TD 5

Énoncé

On se propose de réaliser un mini-embryon de navigateur dans un plan du métro de Paris capable de nous donner un chemin entre deux stations. Il vous a été mentionné en cours que la structure de données permettant de représenter un réseau est le graphe.

1 Navigation entre les stations

Q1 Ce graphe est-il orienté ou non-orienté ?

Nous choisissons donc de le considérer non-orienté. Les données du plan simplifié (pas de ligne 10 et pas de RER) vous sont fournies sous forme textuelle dans le fichier `plan.txt`. La structure de ce fichier est très simple. Une ligne de métro débute par son nom puis vient la liste de toutes les stations qui se suivent jusqu'à un caractère « . » qui marque la fin de la ligne de métro décrite. Chaque élément est séparé par un retour à la ligne (pas de métro !).

Pour référence, il vous est également donné le fichier `plan.jpg` qui est un extrait du plan de métro de la RATP, ce qui vous permettra de voir où se trouvent effectivement les stations du plan simplifié et les connexions entre les lignes (pour tester votre programme).

Q2 Proposez une structure de nœud du graphe, pour représenter une station. Pensez à l'utilisation que nous voulons faire de ce graphe.

2 Création du plan

La structure de données que nous allons utiliser vous est fournie dans le fichier `station.h`. Pour le moment, **ne vous occupez pas** du dernier champ de la structure.

Q3 Quel va être le prototype de la fonction `load_map` chargée de créer le graphe en mémoire à partir d'un fichier de description ?

Pour alléger le travail, il vous est donné, dans le fichier `load_map_skel.(c|h)`, le squelette de la fonction `load_map` que vous allez écrire et 2 fonctions :

- **struct** station_t* get_or_create_station (**char** *name)
  qui prend en argument un nom de station et recherche (dans la liste chaînée globale `all_stations`) cette station ou la crée si elle n'existe pas et retourne un pointeur dessus. Ceci permet d'être certain que les stations existent de manière unique.
- **struct** stations_list_t * get_all_stations (**void**)
  qui retourne la tête de la liste chaînée des stations créées (en fait, retourne la variable globale `all_stations`). Cette fonction permet juste de ne pas laisser visible la variable globale depuis les autres fichiers.

Le but est maintenant d'écrire la fonction qui va charger le plan en mémoire et construire la graphe.

Q4 Quelle va être la forme de l'algorithme de création du graphe ?

Pour lire les noms de station, on n'utilisera **pas** `fscanf` car ces noms peuvent être composés (i.e. avec des espaces). À la place on utilisera la fonction :

char* fgets (**char** *str, **int** size, FILE *in) ;

qui permet de lire une **ligne de texte** dans un fichier. Une ligne de texte est une suite de caractères terminée par un retour chariot (`'\n'`). Cette fonction prend en arguments la zone mémoire où mettre les caractères lus, le nombre maximal de caractères à lire et le descripteur du fichier. Cette fonction retourne `NULL` en fin de fichier (donc principe différent de `fscanf`/`feof`).

Attention Le retour chariot de fin de ligne est stocké dans la chaîne. Il faudra donc penser à le supprimer.

Q5 Écrivez, dans le fichier `load_map_skel.c` (que vous pouvez renommer), la fonction `load_map`.

Dans le fichier `utils.(c|h)` vous sont données 2 fonctions :

- `print_stations` : permet d'afficher toutes les stations de la liste chaînée passée en argument.
- `find_station` permet de rechercher dans une liste chaînée une station par son nom (retourne `NULL` en cas d'échec).

3 Recherche de chemin

Q6 Votre programme devra trouver le chemin **le plus court en nombre de stations** et afficher les stations formant ce chemin. De quelle manière faut-il parcourir le graphe ?

Pour alléger le travail, il vous est donné dans le fichier `queue.(c|h)`, un ensemble de fonctions permettant de créer, manipuler, libérer des **files** de **stations** (**struct** station_t).

```
-------------- queue.h --------------
#ifndef __QUEUE_H__
#define __QUEUE_H__
#include <stdbool.h>
#include "station.h"

struct queue_t {
  unsigned int max_nb ;       /* Nombre max d'éléments. */
  unsigned int cur_nb ;       /* Nombre actuel d'éléments. */
  unsigned int first ;        /* Indice du premier élément. */
  struct station_t **data ;
};

struct queue_t* queue_alloc (unsigned int max_size) ;
void queue_free (struct queue_t *q) ;
struct station_t* take (struct queue_t *q) ;
void enqueue (struct queue_t *q, struct station_t *pi) ;
bool is_empty (struct queue_t *q) ;
#endif
```

Q7 Écrivez la fonction `shortest_path` qui prend en argument 2 pointeurs sur station (1 station de départ et 1 station d'arrivée) et retourne un booléen disant s'il existe un chemin entre ces 2 stations, **en effectuant la recherche du plus court chemin**.

Q5 Que manque-t'il pour être capable d'afficher l'ensemble des stations du chemin **le plus court** trouvé ?

Q6 Modifiez votre algorithme de parcours pour être capable d'afficher l'ensemble des stations du chemin **le plus court** trouvé. Pour simplifier, on pourra s'autoriser à afficher les stations composant le chemin en ordre inverse (donc, destination vers source).

Q7 Si vous avez le temps, écrivez votre `main`, sinon utilisez le fichier `main.c` qui vous est donné (il faut que vous ayez respecté les noms). Testez votre programme.